实 验 报 告 内 容

一・实验内容

题目一：Hadoop 安装使用、HDFS 编程实践、HBase 编程实践、MapReduce 编程实践

实验内容与要求：

1. Hadoop安装使用
    1. 在 PC 机上以伪分布式模式安装 Hadoop；
    2. 访问 Web 界面查看 Hadoop 信息。
2. HDFS 编程实践
    1. 使用 HDFS 文件操作的常用 Shell 命令；
    2. 利用 Hadoop 提供的 Java API 进行基本的文件操作。
3. HBase 编程实践
    1. 在 Hadoop 基础上安装 HBase；
    2. 使用 Shell 命令（`create`: 创建表、`list`: 列出 HBase 所有的表信息、`put`: 向表、行、列指定的单元格添加数据等）；使用 HBase 常用 Java API 创建表、插入数据、浏览数据。
4. MapReduce 编程实践
    1. 使用 MapReduce 实现多个文本文件中 `WordCount` 词频统计功能，实验编写 Map 处理逻辑、编写 Reduce 处理逻辑、编写 `main` 方法。

二・实验目的

1. Hadoop安装使用
    1. 熟悉 Hadoop 的安装流程。
    2. 熟悉 Hadoop 访问 Web 界等基本操作。
2. HDFS 编程实践
    1. 理解 HDFS 在 Hadoop 体系结构中的角色。
    2. 熟练使用 HDFS 操作常用的 Shell 命令。
    3. 熟悉 HDFS 操作常用的 Java API。
3. HBase 编程实践
    1. 理解 HBase 在 Hadoop 体系结构中的角色。
    2. 熟练使用 HBase 操作常用的 Shell 命令。
    3. 熟悉 HBase 操作常用的 Java API。
4. MapReduce 编程实践

1. 通过实验掌握基本的 MapReduce 编程方法。
2. 实现统计 HDFS 系统中多个文本文件中的单词出现频率。

三・实验过程截图及说明

3.1 实验一 Hadoop安装使用

本实验要求：

1. 在 PC 机上以伪分布式模式安装 Hadoop；
2. 访问 Web 界面查看 Hadoop 信息。

3.1.1 在 PC 机上以伪分布式模式安装 Hadoop

本实验在单机上使用 `Docker Compose` 以伪分布式方式部署 Hadoop 相关组件。`docker-compose.yml` 定义 `namenode`、`datanode`、`resourcemanager`、`nodemanager`、`lab-client` 等服务，并通过 `hadoop.env` 配置 `fs.defaultFS`、`dfs.replication` 与 `dfs.permissions.enabled` 等参数。

`docker-compose.yaml` 文件如下：

```yaml
networks:
  bigdata-net:
    driver: bridge

volumes:
  namenode:
  datanode:

services:
  # ----------------------------
  # 1. HDFS 存储层
  # ----------------------------
  namenode:
    image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
    container_name: namenode
    hostname: namenode
    restart: always
    ports:
      - "9870:9870"
      - "9000:9000"
    env_file:
      - ./hadoop.env
    volumes:
      - namenode:/hadoop/dfs/name
    networks:
      - bigdata-net

  datanode:
    image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
    container_name: datanode
    hostname: datanode
    restart: always
    environment:
      SERVICE_PRECONDITION: "namenode:9870 namenode:9000"
    env_file:
      - ./hadoop.env
    volumes:
      - datanode:/hadoop/dfs/data
    networks:
      - bigdata-net

  # ----------------------------
  # 2. YARN 计算层 (MapReduce)
  # ----------------------------
  resourcemanager:
    image: bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8
    container_name: resourcemanager
    hostname: resourcemanager
    restart: always
    ports:
```

```
      - "8088:8088"
    environment:
      SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode:9864"
    env_file:
      - ./hadoop.env
    networks:
      - bigdata-net

  nodemanager:
    image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
    container_name: nodemanager
    hostname: nodemanager
    restart: always
    environment:
      SERVICE_PRECONDITION: "namenode:9000 namenode:9870 datanode:9864
resourcemanager:8088"
    env_file:
      - ./hadoop.env
    networks:
      - bigdata-net

  # ----------------------------
  # 3. Zookeeper & HBase
  # ----------------------------
  zookeeper:
    image: zookeeper:3.8
    container_name: zookeeper
    hostname: zookeeper
    restart: always
    ports:
      - "2181:2181"
    networks:
      - bigdata-net

  hbase:
    image: harisekhon/hbase:latest
    container_name: hbase
    hostname: hbase
    restart: always
    ports:
      - "16010:16010"
    depends_on:
      - namenode
      - zookeeper
    environment:
      - HBASE_MANAGES_ZK=false
      - HBASE_ZOOKEEPER_QUORUM=zookeeper
      - HBASE_ROOTDIR=hdfs://namenode:9000/hbase
      - SERVICE_PRECONDITION=namenode:9000 zookeeper:2181
    networks:
      - bigdata-net
```

```yaml
  # ----------------------------
  # 4. 全能工作台 (Client)
  # ----------------------------
  client:
    image: bde2020/hadoop-base:2.0.0-hadoop3.2.1-java8
    container_name: lab-client
    hostname: client
    restart: always
    volumes:
      - ./workspace:/workspace
    env_file:
      - ./hadoop.env
    networks:
      - bigdata-net
    environment:
      CORE_CONF_fs_defaultFS: hdfs://namenode:9000
      YARN_CONF_yarn_resourcemanager_hostname: resourcemanager
      SERVICE_PRECONDITION: "namenode:9000 resourcemanager:8088 hbase:16010"
    command: tail -f /dev/null
```

`hadoop.env` 文件如下：

```
CLUSTER_NAME=test
CORE_CONF_fs_defaultFS=hdfs://namenode:9000
CORE_CONF_hadoop_http_staticuser_user=root
CORE_CONF_hadoop_proxyuser_hue_hosts=*
CORE_CONF_hadoop_proxyuser_hue_groups=*
HDFS_CONF_dfs_webhdfs_enabled=true
HDFS_CONF_dfs_permissions_enabled=false
HDFS_CONF_dfs_replication=1
YARN_CONF_yarn_log_aggregation_enable=true
YARN_CONF_yarn_resourcemanager_recovery_enabled=true
YARN_CONF_yarn_resourcemanager_store_class=org.apache.hadoop.yarn.server.resourcemanag
er.recovery.FileSystemRMStateStore
YARN_CONF_yarn_resourcemanager_fs_state_store_uri=/rmstate
YARN_CONF_yarn_nodemanager_remote_app_log_dir=/app-logs
YARN_CONF_yarn_log_aggregation_retain_seconds=-1
YARN_CONF_yarn_log_aggregation_retain_check_interval_seconds=-1
```

启动命令如下（在宿主机执行）：

```
docker compose up -d
docker exec -it lab-client bash
```

3.1.2 访问 Web 界面查看 Hadoop 信息

启动后分别访问以下页面查看 Hadoop 信息：

1. HDFS 管理界面：`http://localhost:9870`（查看 DataNode 信息）
2. YARN 管理界面：`http://localhost:8088`（查看集群资源）
3. HBase 管理界面：`http://localhost:16010`（供实验三查看 RegionServer）

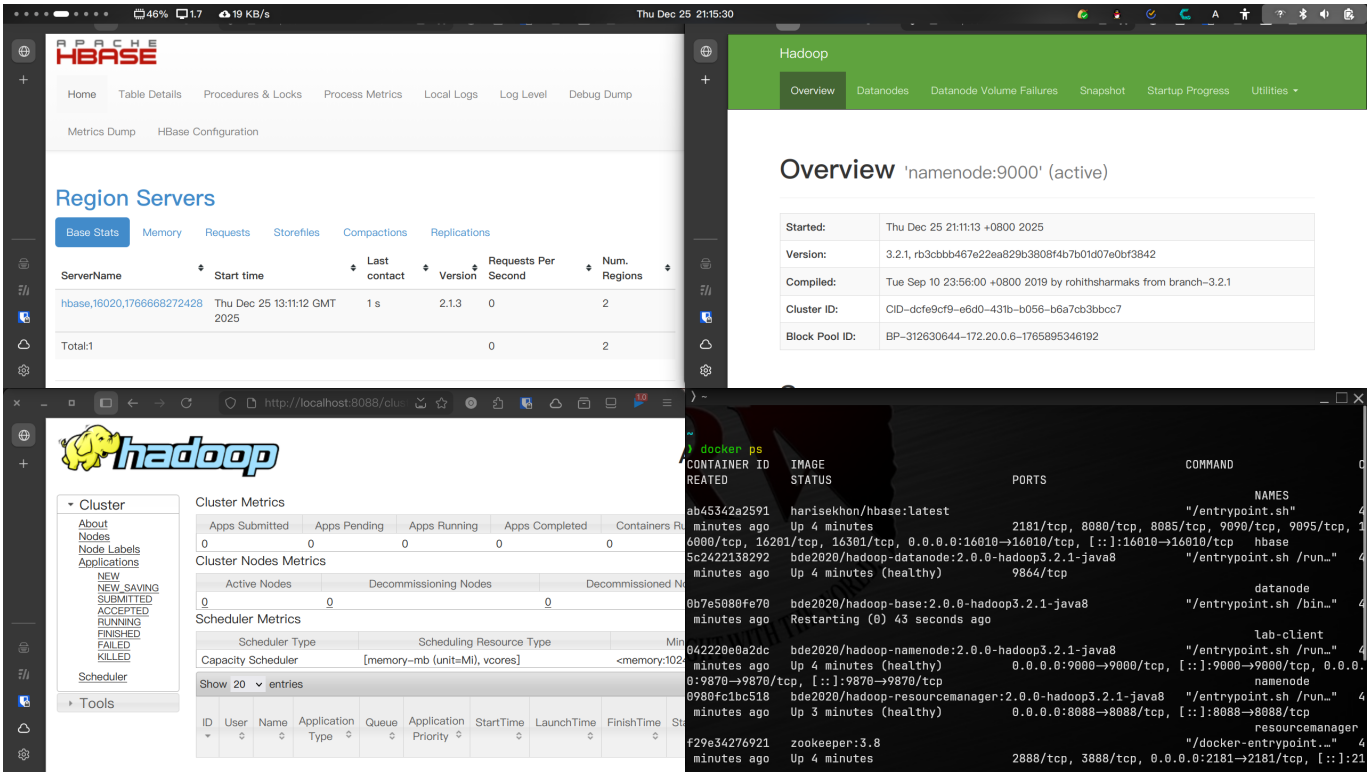

图 1 Hadoop/HDFS/YARN/HBase Web 界面与容器状态

3.2 实验二 HDFS 编程实践

本实验要求：

1. 使用 HDFS 文件操作的常用 Shell 命令；
2. 利用 Hadoop 提供的 Java API 进行基本的文件操作。

3.2.1 使用 HDFS 文件操作的常用 Shell 命令

在 `lab-client` 容器中进入挂载目录 `/workspace` 后，执行 HDFS 常用命令完成目录创建、文件上传与查看：

```
cd /workspace
hdfs dfs -mkdir -p /user/root/input
echo "Hello CachyOS Hello Hadoop" > test.txt
hdfs dfs -put test.txt /user/root/input/
hdfs dfs -ls /user/root/input/
hdfs dfs -cat /user/root/input/test.txt
```

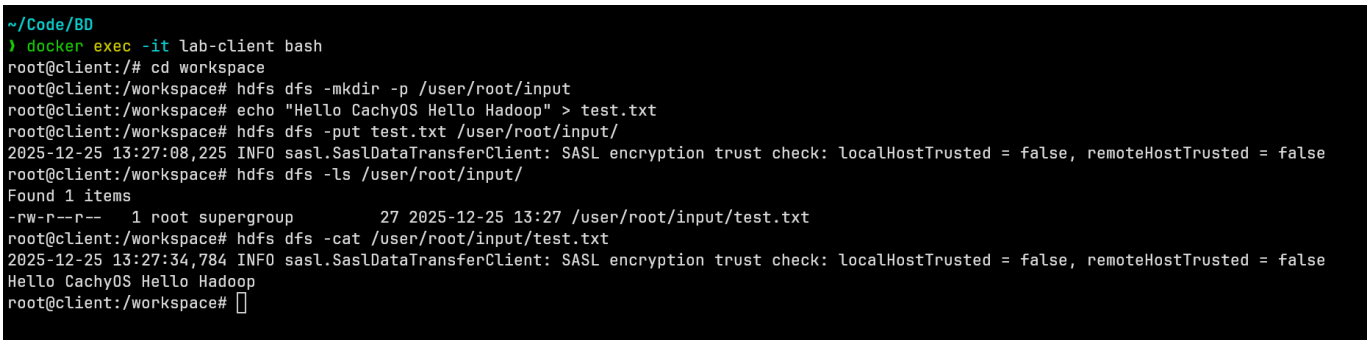

图 2 HDFS Shell 命令操作与输出

3.2.2 利用 Hadoop 提供的 Java API 进行基本的文件操作

编写并运行 `HDFSTest.java`，使用 Hadoop Java API 对 HDFS 执行一组可重复的文件操作流程，覆盖如下操作：

1. `Create`：创建文件并写入内容。
2. `Read`：读取文件内容并输出。
3. `Rename`：以重命名方式模拟更新操作。
4. `List`：读取文件状态信息（路径、属主、大小、副本数等）。
5. `Delete`：删除测试文件并清理环境。

其中，HDFS 访问地址使用 `hdfs://namenode:9000`，测试文件路径为 `/user/root/input/java_crud_test.txt`，重命名后的路径为 `/user/root/input/java_crud_renamed.txt`。

编译与运行命令如下（在容器内执行）：

```
javac -cp `hadoop classpath` HDFSTest.java
java -cp .:`hadoop classpath` HDFSTest
```

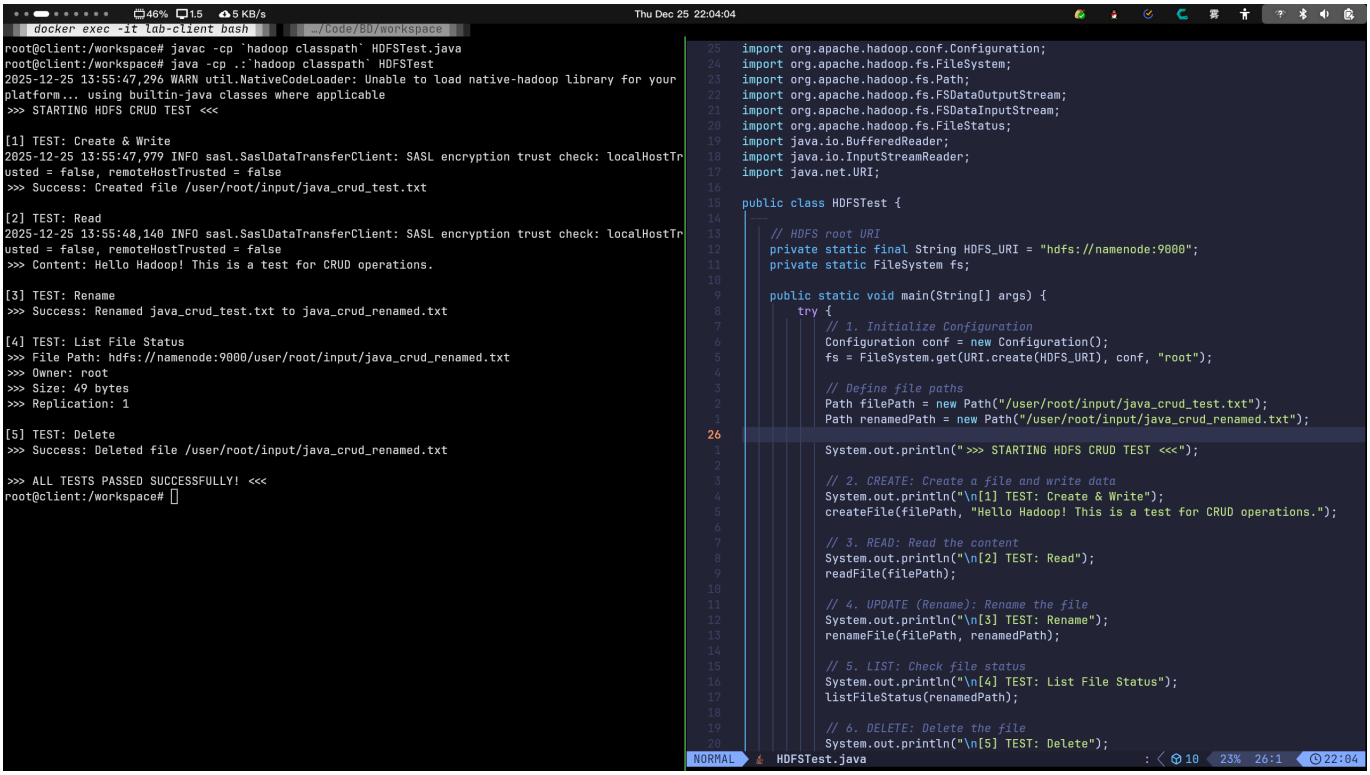

图 3 HDFS Java API（CRUD）代码与运行结果

3.3 实验三 HBase 编程实践

本实验要求：

1. 在 Hadoop 基础上安装 HBase；
2. 使用 Shell 命令（`create`: 创建表、`list`: 列出 HBase 所有的表信息、`put`: 向表、行、列指定的单元格添加数据等）；使用 HBase 常用 Java API 创建表、插入数据、浏览数据。

3.3.1 在 Hadoop 基础上安装 HBase

HBase 与 Zookeeper 在 `Docker Compose` 中一并部署，HBase 通过配置将 `HBase RootDir` 指向 HDFS（`hdfs://namenode:9000/hbase`）。HBase Web UI 可通过 `http://localhost:16010` 访问。

3.3.2 使用 HBase Shell 命令

进入 `hbase shell`（例如执行 `docker exec -it hbase hbase shell`）后，完成 `DDL`/`DML`/查询/删除与删表等操作。

3.3.2.1 Namespace 与建表（DDL）

查看状态、创建表 `student`（列族 `info` 与 `score`）、查看表结构并列出表：

```
status 'detailed'
create 'student', 'info', 'score'
describe 'student'
list
```

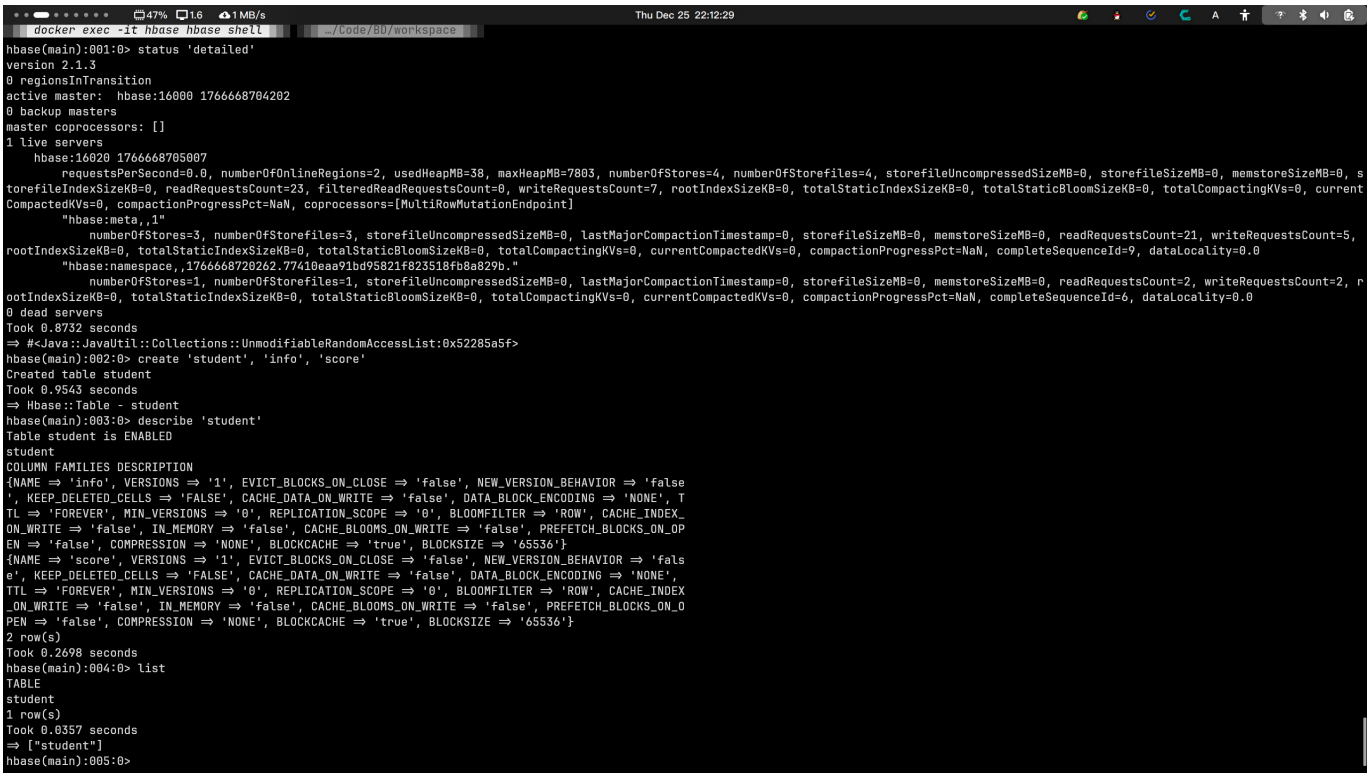

图 4 HBase Shell：状态查看与建表

3.3.2.2 数据插入

以学号作为 `RowKey`，向 `info` 与 `score` 列族写入不同列的数据：

```
put 'student', '1001', 'info:name', 'Alice'
put 'student', '1001', 'info:class', 'CS-1'
put 'student', '1001', 'score:math', '85'
put 'student', '1001', 'score:english', '90'

put 'student', '1002', 'info:name', 'Bob'
put 'student', '1002', 'info:class', 'CS-2'
put 'student', '1002', 'score:math', '92'
```

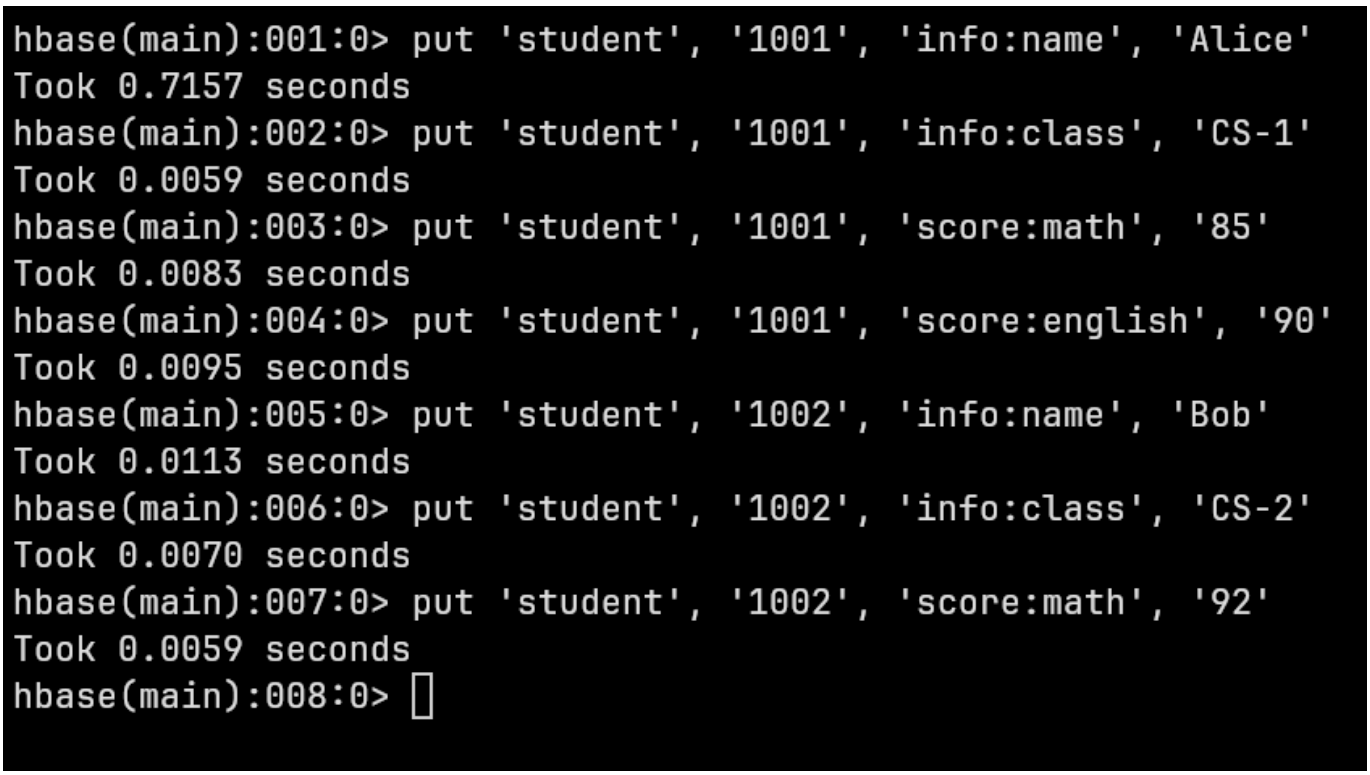

图 5 HBase Shell：数据插入

3.3.2.3 数据查询

通过 `scan` 查看全表，通过 `get` 查询指定行或指定列族/列：

```
scan 'student'
get 'student', '1001'
get 'student', '1001', 'info:name'
get 'student', '1002', 'score'
```

```
docker exec -it hbase hbase shell    …/Code/BD/workspace
hbase(main):008:0> scan 'student'
ROW                                             COLUMN+CELL
 1001                                           column=info:class, timestamp=1766672078767, value=CS-1
 1001                                           column=info:name, timestamp=1766672073205, value=Alice
 1001                                           column=score:english, timestamp=1766672087429, value=90
 1001                                           column=score:math, timestamp=1766672083399, value=85
 1002                                           column=info:class, timestamp=1766672095523, value=CS-2
 1002                                           column=info:name, timestamp=1766672091380, value=Bob
 1002                                           column=score:math, timestamp=1766672101605, value=92
2 row(s)
Took 0.0642 seconds
hbase(main):009:0> get 'student', '1001'
COLUMN                                          CELL
 info:class                                     timestamp=1766672078767, value=CS-1
 info:name                                      timestamp=1766672073205, value=Alice
 score:english                                  timestamp=1766672087429, value=90
 score:math                                     timestamp=1766672083399, value=85
1 row(s)
Took 0.0571 seconds
hbase(main):010:0> get 'student', '1001', 'info:name'
COLUMN                                          CELL
 info:name                                      timestamp=1766672073205, value=Alice
1 row(s)
Took 0.0119 seconds
hbase(main):011:0> get 'student', '1002', 'score'
COLUMN                                          CELL
 score:math                                     timestamp=1766672101605, value=92
1 row(s)
Took 0.0152 seconds
hbase(main):012:0>
```

图 6 HBase Shell：数据查询

3.3.2.4 数据删除与清空

删除单元格后再次查询验证；随后删除整行数据：

```
delete 'student', '1002', 'score:math'
get 'student', '1002'
deleteall 'student', '1002'
```

```
docker exec -it hbase hbase shell    …/Code/BD/workspace
hbase(main):012:0> delete 'student', '1002', 'score:math'
Took 0.0166 seconds
hbase(main):013:0> get 'student', '1002'
COLUMN                                          CELL
 info:class                                     timestamp=1766672095523, value=CS-2
 info:name                                      timestamp=1766672091380, value=Bob
1 row(s)
Took 0.0129 seconds
hbase(main):014:0> deleteall 'student', '1002'
Took 0.0064 seconds
hbase(main):015:0>
```

图 7 HBase Shell：数据删除与验证

3.3.2.5 删表操作

删除表前先 `disable`：

```
disable 'student'
drop 'student'
list
```

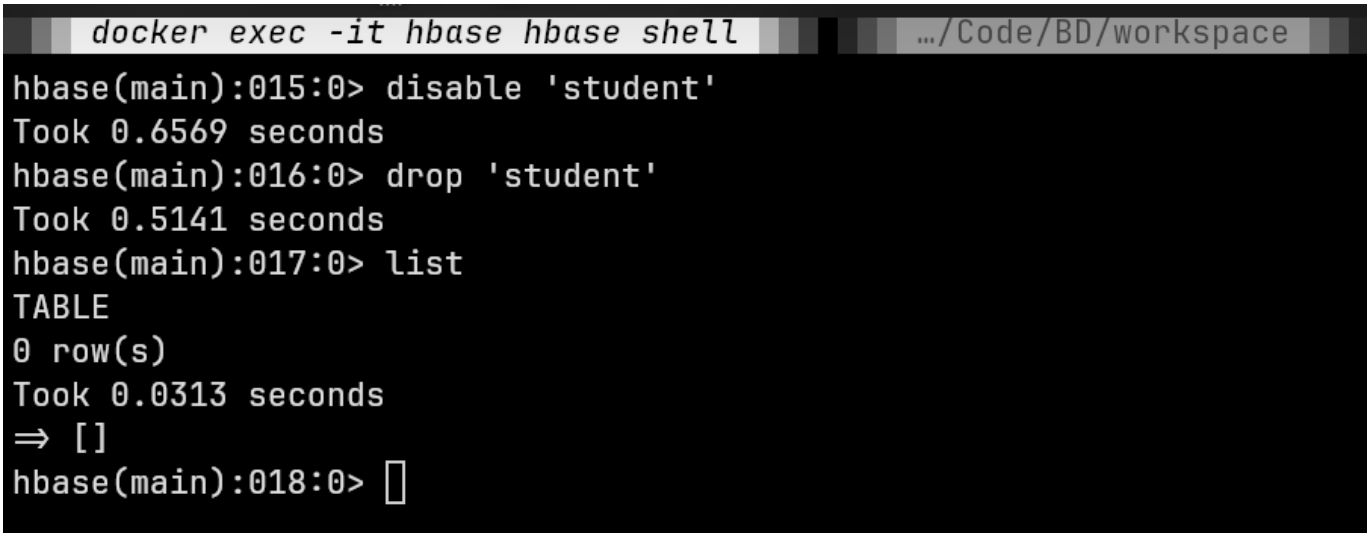

图 8 HBase Shell：删表操作

3.3.3 使用 HBase 常用 Java API 创建表、插入数据、浏览数据

编写 `HBaseTest.java`，通过 Java API 完成连接、建表、插入与扫描读取流程，关键步骤如下：

1. 构建 `HBaseConfiguration`，并指定 `zookeeper` 与端口。
2. 创建连接与 `Admin`，检查表是否存在，存在则删除后重建以保证测试可重复。
3. 创建表 `student_java`（列族 `info`），写入 `RowKey` =1001 的两列数据（`name`、`os`）。
4. `Scan` 全表并输出读取到的列值。

编译与运行命令如下（在容器内执行）：

```
javac -cp .:$(hadoop classpath):/workspace/hbase-lib/* HBaseTest.java
java -cp .:$(hadoop classpath):/workspace/hbase-lib/* HBaseTest
```

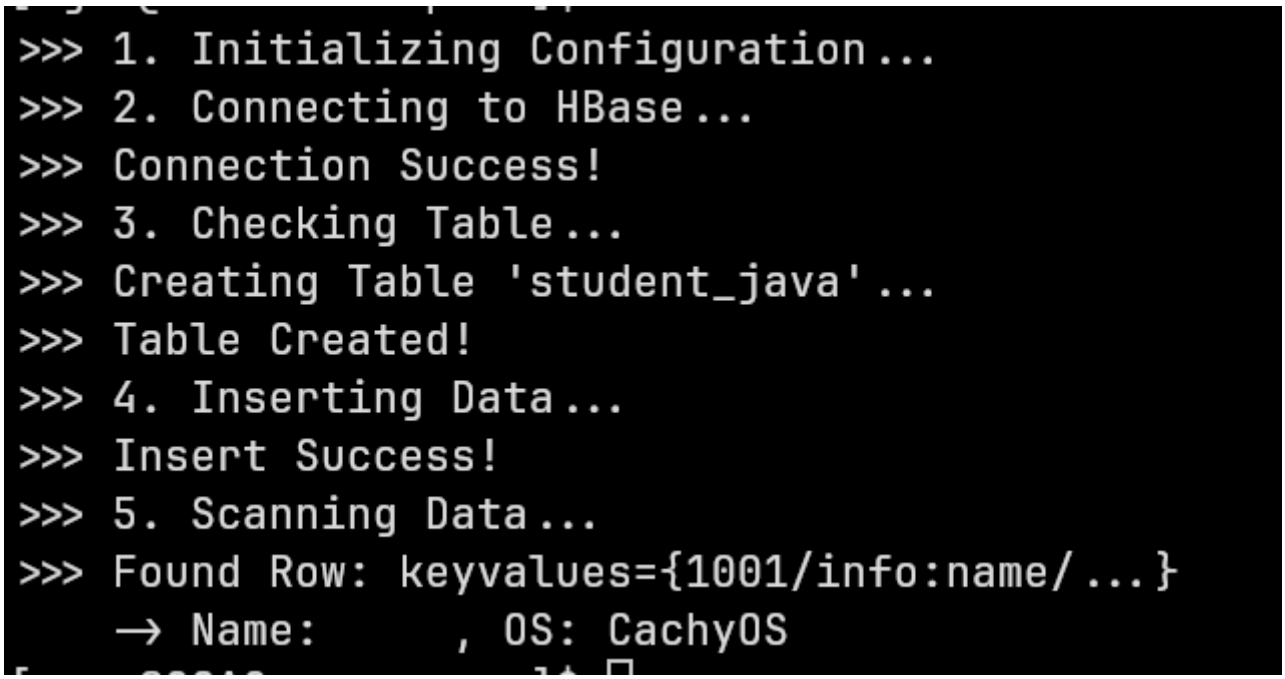

图 9 HBase Java API：建表、写入与扫描

3.4 实验四 MapReduce 编程实践

本实验要求：

使用 MapReduce 实现多个文本文件中 `WordCount` 词频统计功能，实验编写 Map 处理逻辑、编写 Reduce 处理逻辑、编写 `main` 方法。

3.4.1 生成数据

在容器中准备 3 个文本文件并上传到 HDFS 输入目录 `/input`：

```
echo "Hello Hadoop Hello MapReduce" > file1.txt
echo "MapReduce is cool and Hadoop is powerful" > file2.txt
echo "CachyOS runs Docker very fast" > file3.txt

hdfs dfs -mkdir -p /input
hdfs dfs -put file1.txt file2.txt file3.txt /input/
hdfs dfs -ls /input/
```

3.4.2 编写 `WordCount` 程序

实现 `TokenizerMapper`（分词并输出 `<word, 1>`）、`IntSumReducer`（对同一 `word` 聚合求和），并在 `main` 方法中配置 `Job` 的输入输出路径与类信息。

核心代码如下：

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text,
IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer extends
Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
```

```
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

3.4.3 编译、打包与运行

编译与打包：

```
rm -f *.class
javac -cp `hadoop classpath` WordCount.java
jar cf wc.jar WordCount*.class
```

运行任务（注意输出目录必须不存在）：

```
hadoop jar wc.jar WordCount /input /output
```

3.4.4 查看输出结果

查看输出目录并读取结果文件：

```
hdfs dfs -ls /output
hdfs dfs -cat /output/part-r-00000
```

词频统计输出如下：

```
CachyOS 1
Docker 1
Hadoop 2
Hello 2
MapReduce 2
and 1
cool 1
fast 1
is 2
powerful 1
runs 1
very 1
```

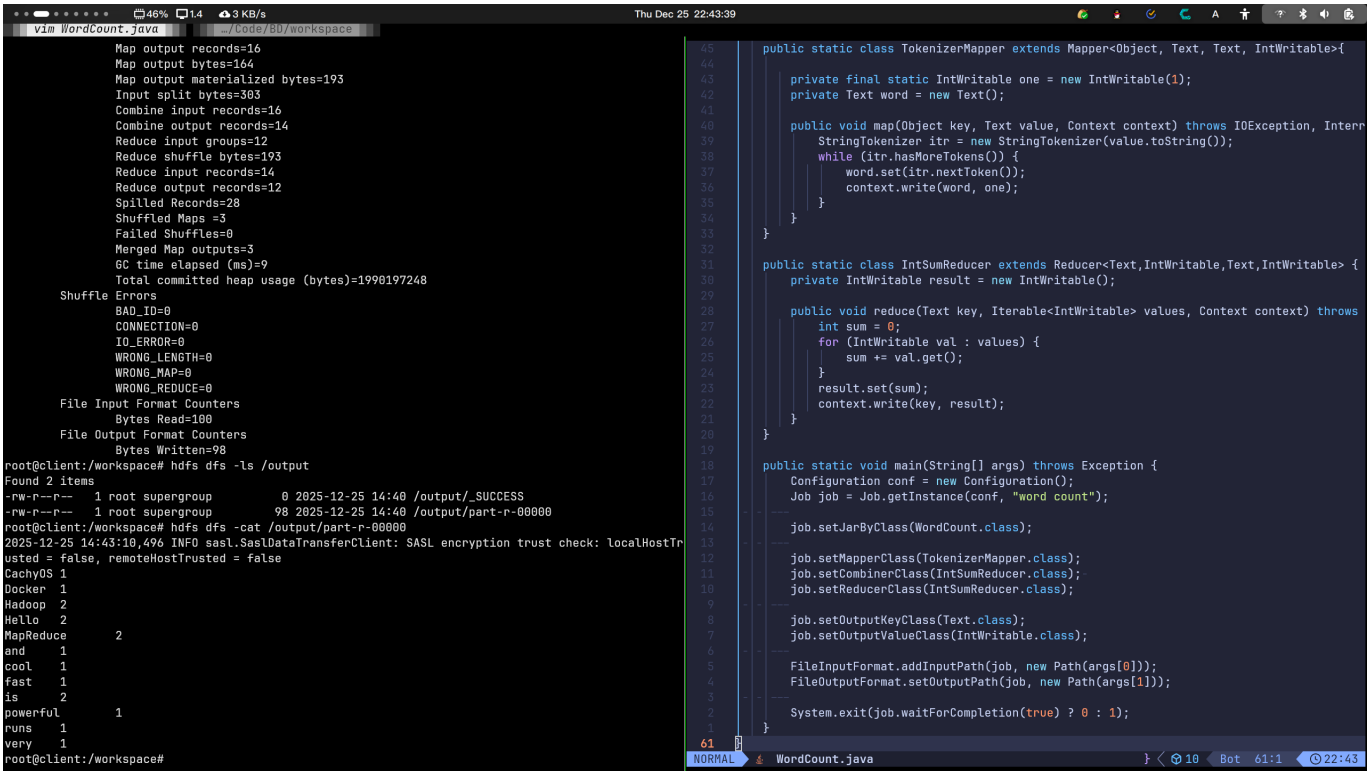

图 10 MapReduce WordCount：运行与输出结果

四·实验总结及心得体会

1. 本次实验通过 `Docker Compose` 部署 `namenode`、`datanode` 与 YARN 相关组件，并通过 `http://localhost:9870`、`http://localhost:8088` 等 Web 界面验证服务状态，完成 Hadoop 安装使用部分的要求。
2. HDFS 编程实践同时覆盖 Shell 与 Java API 两条路径。Shell 侧完成 `hdfs dfs -mkdir`、`hdfs dfs -put`、`hdfs dfs -ls`、`hdfs dfs -cat` 等基础操作；Java 侧通过 `FileSystem` 与 `Path` 等 API 完成创建、读取、重命名、状态查询与删除，验证了基本文件操作链路。
3. HBase 编程实践在 Shell 侧完成建表、写入、查询、删除与删表；在 Java API 侧完成连接、建表、写入与扫描读取，验证了程序访问 HBase 的基本流程。
4. MapReduce 编程实践完成 `WordCount` 的 `Mapper`、`Reducer` 与 `Driver` 编写，并在 HDFS 输入输出目录 `/input` 与 `/output` 上完成任务运行与结果验证。